



如何角逐云计算市场？  
老程序员带你解析Python多线程  
大数据时代，到底大在哪里？  
Linux运维工程师的“小秘密”

**51CTO** | 开发频道 首页 Web 架构&设计 语言&工具 大数据

输入您要搜索的内容

## 走近Node.js的异步代码设计

许多企业目前在评估Node.js的异步、事件驱动型的I/O，认为这是一种高性能方案，可以替代多线程企业应用服务器的传统同步I/O。异步性质意味着，企业开发人员必须学习新的编程模式，忘掉旧的编程模式。他们必须彻底转变思路，可能需要借助电击疗法^\_^。本文介绍了如何将旧的同步编程模式换成全新的异步编程模式。

作者：布加迪编译 来源：51CTO | 2011-12-23 13:58

移动端

收藏

分享

**Tech Neo技术沙龙 | 11月25号，九州云/ZStack与您一起探讨云时代网络边界管理实践**

【51CTO精选译文】许多企业目前在评估Node.js的异步、事件驱动型的I/O，认为这是一种高性能方案，可以替代多线程企业应用服务器的传统同步I/O。异步性质意味着，企业开发人员必须学习新的编程模式，忘掉旧的编程模式。他们必须彻底转变思路，可能需要借助电击疗法^\_^。本文介绍了如何将旧的同步编程模式换成全新的异步编程模式。

51CTO推荐专题：[Node.js专区](#)

开始转变思路



### 编辑推荐

热点

惹毛程序员的十件事！需求变更居然不是排第一！

头条

高性能Java持久化的14个技巧

要使用Node.js，就有必要了解异步编程的工作原理。异步代码设计并非简单的设计，需要一番学习。现在需要来一番电击疗法：本文在同步代码示例旁边给出了异步代码示例，表明如何更改同步代码，才能变成异步代码。这些示例都围绕Node.js的文件系统(fs)模块，因为它是唯一含有同步I/O操作及异步I/O操作的模块。有了这两种示例，你可以开始转变思路了。

## 相关代码和独立代码

回调函数(callback function)是Node.js中异步事件驱动型编程的基本构建模块。它们是作为变量，传递给异步I/O操作的函数。一旦操作完成，回调函数就被调用。回调函数是Node.js中实现事件的机制。

下面显示的示例表明了如何将同步I/O操作转换成异步I/O操作，并显示了回调函数的使用。示例使用异步fs.readdirSync()调用，读取当前目录的文件名称，然后把文件名称记录到控制台，最后读取当前进程的进程编号(process id)。

### 同步

```
1. var fs = require('fs'),
2.     filenames,
3.     i,
4.     processId;
5. filenames = fs.readdirSync(".");
6. for (i = 0; i < filenames.length; i++) {
7.     console.log(filenames[i]);
8. }
9. console.log("Ready.");
10. processprocessId = process.getuid();
```

热点 号称世界最快句法分析器，Python高级自然语言处理库spaCy！

头条 跨界转行做编程的5大女神，新一代码农女神在谷歌做实习生！

头条 腾讯面试官送给准程序员的一些建议！

## 24H热文 一周话题 本月最赞

坐在马桶上看算法：快速排序

Java程序员新手老手都离不开八大开发工具

5个强大的Java分布式缓存框架推荐

二维码的生成细节和原理

Java 中常用缓存Cache机制的实现

成为Java顶尖程序员，看这11本书就够了

我用Python爬了7W知乎用户信息，终于捕...

挨踢部落坐诊第十一期：三千万数据如何做...



视频课程

+更多

## 异步

```
1. var fs = require('fs'),
2.    processId;
3. fs.readdir(".", function (err, filenames) {
4.     var i;
5.     for (i = 0; i < filenames.length; i++) {
6.         console.log(filenames[i]);
7.     }
8.     console.log("Ready.");
9. });
10. processprocessId = process.getuid();
```

在同步示例中，处理器等待fs.readdirSync() I/O操作，所以这是需要更改的操作。Node.js中该函数的异步版本是fs.readdir()。它与fs.readdirSync()一样，但是回调函数作为第二个参数。

使用回调函数模式的规则如下：把同步函数换成对应的异步函数，然后把原先在同步调用后执行的代码放在回调函数里面。回调函数中的代码与同步示例中的代码执行一模一样的操作。它把文件名称记录到控制台。它在异步I/O操作返回之后执行。

就像文件名称的记录依赖fs.readdirSync() I/O操作的结果，所列文件数量的记录也依赖其结果。进程编号的存储独立于I/O操作的结果。因而，必须把它们移到异步代码中的不同位置。

规则就是将相关代码移到回调函数中，而独立代码的位置不用管。一旦I/O操作完成，相关代码就被执行，而独立代码在I/O操作被调用之后立即执行。

## 顺序



Juniper入门与提高实战视频课程【大侠唐在飞

讲师：大侠唐在飞 30835人学习过



【范昌明】从零开始系列-Project 2010视频课

讲师：范昌明 132020人学习过



这可能是你见过“最牛逼”的C++课程

讲师：王桂林 160393人学习过

## 最新专题

[+更多](#)

未来即将“触脸可及”，人脸识别技术大揭秘！

未来



关于智能运维的探索与实践

智能运维



智慧城市的背后是与前沿技术的深度挖掘和利用

智慧城市

同步代码中的标准模式是线性顺序：几行代码都必须下一行接上一行来执行，因为每一行代码依赖上一行代码的结果。在下面示例中，代码首先变更了文件的访问模式(比如Unix `chmod`命令)，对文件更名，然后检查更名后文件是不是符号链接。很显然，该代码无法乱序运行，不然文件在模式变更前就被更名了，或者符号链接检查在文件被更名前就执行了。这两种情况都会导致出错。因而，顺序必须予以保留。

## 同步

```
1. var fs = require('fs'),
2.   oldFilename,
3.   newFilename,
4.   isSymLink;
5. oldFilename = "./processId.txt";
6. newFilename = "./processIdOld.txt";
7. fs.chmodSync(oldFilename, 777);
8. fs.renameSync(oldFilename, newFilename);
9. isSymLink = fs.lstatSync(newFilename).isSymbolicLink();
```

## 异步

```
1. var fs = require('fs'),
2.   oldFilename,
3.   newFilename;
4. oldFilename = "./processId.txt";
5. newFilename = "./processIdOld.txt";
6. fs.chmod(oldFilename, 777, function (err) {
7.   fs.rename(oldFilename, newFilename, function (err) {
8.     fs.lstat(newFilename, function (err, stats) {
9.       var isSymLink = stats.isSymbolicLink();
10.    });
11.  });
```



HTML5游戏开发难点之效率、性能和加载量

HTML5游戏

## 精彩评论



漏水亦凡评论了：【IT观察】从全球女鹿饭同时失联引发的悲剧看运维如何在流量峰值力挽狂澜

分析还可以



dtchong评论了：1024，给你10000个不加班的理由！

我澄清，我是个不戴眼镜的程序媛，虽然戴眼镜会显得(๑˙˙˙)σ有学问



westwindluwei评论了：华为离职副总裁徐家骏：透露年薪千万的工作感悟，太震撼了！

为敢想敢干的人点赞



ttxs122评论了：【IT观察】从全球女鹿饭同时失联引发的悲剧看运维如何在流量峰值力挽狂澜

分析很到位，对我帮助很大

```
11.   });  
12.   });
```

在异步代码中，这些顺序变成了嵌套回调。该示例显示了fs.lstat()回调嵌套在fs.rename()回调里面，而fs.rename()回调嵌套在fs.chmod()回调里面。

## 并行处理

异步代码特别适合操作I/O操作的并行处理：代码的执行并不因I/O调用的返回而受阻。多个I/O操作可以并行开始。在下面示例中，某个目录中所有文件的大小都在循环中累加，以获得那些文件占用的总字节数。使用异步代码，循环的每次迭代都必须等到获取单个文件大小的I/O调用返回为止。

异步代码允许快速连续地在循环中开始所有I/O调用，不用等结果返回。只要其中一个I/O操作完成，回调函数就被调用，而该文件的大小就可以添加到总字节数中。

唯一必不可少的有一个恰当的停止标准，它决定着我们完成处理后，就计算所有文件的总字节数。

## 同步

```
1.  var fs = require('fs');  
2.  function calculateByteSize() {  
3.      var totalBytes = 0,  
4.          i,  
5.          filenames,  
6.          stats;  
7.      filenames = fs.readdirSync(".");  
8.      for (i = 0; i < filenames.length; i++) {
```

## 精选博文 论坛热帖 下载排行

2011 ARM技术研讨会杂记

跨ESXi主机的裸磁盘映射(RDM)搭建MSC

用Jquery写tab插件(支持点击和移动及

我离职的动机

计算机领域里的“兴趣广”和“无兴趣

## 读书

[+更多](#)

### 征服Python——语言基础与典型应用

Python是目前流行的脚本语言之一。本书由浅入深、循序渐进地讲解如何使用Python进行程序开发。全书内容包括Python安装、开发工具简介、Pyth...



## 订阅51CTO邮刊

[点击这里查看样刊](#)



立即订阅

```
9.     stats = fs.statSync("./" + filenames[i]);
10.     totalBytes += stats.size;
11. }
12. console.log(totalBytes);
13. }
14.
15.
16.
17. calculateByteSize();
```

## 异步

```
1. var fs = require('fs');
2. var count = 0,
3.     totalBytes = 0;
4. function calculateByteSize() {
5.     fs.readdir(".", function (err, filenames) {
6.         var i;
7.         count = filenames.length;
8.         for (i = 0; i < filenames.length; i++) {
9.             fs.stat("./" + filenames[i], function (err, stats) {
10.                 totalBytes += stats.size;
11.                 count--;
12.                 if (count === 0) {
13.                     console.log(totalBytes);
14.                 }
15.             });
16.         }
17.     });
18. }
19. calculateByteSize();
```



同步示例简单又直观。在异步版本中，第一个`fs.readdir()`被调用，以读取目录中的文件名。在回调函数中，针对每个文件调用`fs.stat()`，返回该文件的统计信息。这部分不出所料。

值得关注的方面出现在计算总字节数的`fs.stat()`回调函数中。所用的停止标准是目录的文件数量。变量`count`以文件数量来初始化，倒计数回调函数执行的次数。一旦数量为0，所有I/O操作都被回调，所有文件的总字节数被计算出来。计算完毕后，字节数可以记录到控制台。

异步示例有另一个值得关注的特性：它使用闭包(closure)。闭包是函数里面的函数，内层函数访问外层函数中声明的变量，即便在外层函数已完成之后。`fs.stat()`回调函数是闭包，因为它早在`fs.readdir()`回调函数完成后，访问在该函数中声明的`count`和`totalBytes`这两个变量。闭包有关于它自己的上下文。在该上下文中，可以放置在函数中访问的变量。

要是没有闭包，`count`和`totalBytes`这两个变量都必须是全局变量。这是由于`fs.stat()`回调函数没有放置变量的任何上下文。`calculateBiteSize()`函数早已结束，只有全局上下文仍在那里。这时候闭包就能派得上用场。变量可以放在该上下文中，那样可以从函数里面访问它们。

## 代码复用

代码片段可以在JavaScript中复用，只要把代码片段包在函数里面。然后，可以从程序中的不同位置调用这些函数。如果函数中使用了I/O操作，那么改成异步代码时，就需要某种重构。

下面的异步示例显示了返回某个目录中文件数量的函数`countFiles()`。`countFiles()`使用I/O操作 `fs.readdirSync()` 来确定文件数量。 `span style="font-family: courier new,courier;">countFiles()`本身被调用，使用两个不同的输入参数：

## 同步

```
1. var fs = require('fs');
2. var path1 = "./",
3.     path2 = "../";
4. function countFiles(path) {
5.     var filenames = fs.readdirSync(path);
6.     return filenames.length;
7. }
8. console.log(countFiles(path1) + " files in " + path1);
9. console.log(countFiles(path2) + " files in " + path2);
```

## 异步

```
1. var fs = require('fs');
2. var path1 = "./",
3.     path2 = "../",
4.     logCount;
5. function countFiles(path, callback) {
6.     fs.readdir(path, function (err, filenames) {
7.         callback(err, path, filenames.length);
8.     });
9. }
10. logCount = function (err, path, count) {
11.     console.log(count + " files in " + path);
12. };
13. countFiles(path1, logCount);
14. countFiles(path2, logCount);
```



把fs.readdirSync()换成异步fs.readdir()迫使闭包函数cntFiles()也变成异步，因为调用cntFiles()的代码依赖该函数的结果。毕竟，只有fs.readdir()返回后，结果才会出现。这导致了cntFiles()重构，以便还能接受回调函数。整个控制流程突然倒过来了：不是console.log()调用cntFiles()，cntFiles()再调用fs.readdirSync()，在异步示例中，而是cntFiles()调用fs.readdir()，然后cntFiles()再调用console.log()。

## 结束语

本文着重介绍了异步编程的一些基本模式。将思路转变到异步编程绝非易事，需要一段时间来适应。虽然难度增加了，但是获得的回报是显著提高了并发性。结合JavaScript的快速周转和易于使用等优点，Node.js中的异步编程有望在企业应用市场取得进展，尤其是在新一代高度并发性的Web 2.0应用程序方面。

原文：<http://shinetech.com/thoughts/thought-articles/139-asynchronous-code-design-with-nodejs>

### 【编辑推荐】

1. [使用Node.js开发多人玩的HTML 5游戏](#)
2. [Node.js提速指南](#)
3. [Node.js专区](#)
4. [什么是Node.js？](#)
5. [使用node.js进行服务器端JavaScript编程](#)

【责任编辑：陈贻新 TEL：( 010 ) 68476606】

点赞 0

node.js

分享:

内容点评 已有 0 条评论, 0 次赞

还可以输入500字

请输入你的评论

您还没有登录！请先 [登录](#) 或 [注册](#)

提交

还没有评论内容

大家都在看 猜你喜欢



新浪微博胡南炜：深度学习在微博信息流推荐中的实践



甲骨文云平台全面升级，助力客户稳步上云



迅雷战略转型，开创“共享计算”时代



微软技术暨生态大会，纳德拉为小冰点赞

---

**51CTO旗下网站：** 领先的IT技术网站 51CTO | 领先的中文存储媒体 WatchStor | 中国首个CIO网站 CIOage | 中国首家数字医疗网站 HC3i

---

Copyright©2005-2017 51CTO.COM 版权所有 未经许可 请勿转载